Automated Security Test by using Formal Threat Models on Leakage Detection

P. Santhosh Reddy[1] G.Sireesha[2]

Software Engineer
Marlabs Software (P) Ltd. Bangalore, Karnataka.

[1]santhosh.peddireddy@gmail.com

Abstract: **The data distributors work is to give sensitive data to a set of presumably trusted third party agents. The data i.e., sent to these third parties are available on the unauthorized places like web and or some ones systems, due to data leakage. The distributor must know the way the data was leaked from one or more agents instead of as opposed to having been independently gathered by other means. Our new proposal on data allocation strategies will improve the probability of identifying leakages along with Security attacks typically result from unintended behaviors or invalid inputs. Due to too many invalid inputs in the real world programs is labor intensive about security testing. The most desirable thing is to automate or partially automate security-testing process. In this paper we represented Predicate/ Transition nets approach for security tests automated generation by using formal threat models.**

Key words: Data leakage, Data privacy, Allocation strategies, security testing, Software security.

INTRODUCTION

Our consideration about applications where the Original sensitive data can't be perturbed. For this the application must detect when the sensitive data got leaked and if possible from where. Before sending the sensitive data to the agents we need to convert it to low sensitive data by using the most useful technique perturbation. For example, one can add random noise to certain attributes, or one can replace exact values by ranges [1]. In some critical cases there is no need of altering the original distributor's sensitive data. For example, the alteration of bank account number and salary information not needed if an outsourcer is doing our payroll. In medical field also the researchers need accurate information about patients. Traditionally, leakage detection is handled by water -marking, e.g., a unique code is embedded in each distributed copy. The leaker can identify easily in the hands of the unauthorized party where the copy is discovered later. In some cases this watermarks technique also useful, but it involves some modification of the original sensitive data. If the data recipient is malicious the watermarks can also be destroyed. In this paper, the leakage of a set of objects or records will detect by using the study unobtrusive techniques. The following is the special scenario we will study: the distributor founds same objects in unauthorized place those are distributed to the agents, after distribution. By this moment, the distributor will acquires the information about leaked data where it came from either one or more agents, neither gathered independently by other means. For example a cookie is stolen from a cookie jar, if distributor catches Ram with a single cookie; he can argue that a friend gave him the cookie. But if we catch Ram with five cookies, it is too hard to him to argue that his hands were not in the cookie jar. In (Stanford, 2008) If the distributor sees "enough evidence" that an agent leaked data, he may stop doing business with him, or may initiate legal proceedings.

Security testing needs to target the "presence of an intelligent adversary bent on breaking the system" [10]. The threat model will provide a basis for effective security testing because threat models describe security threats from the standpoint of how the adversary would attack or exploit a system. Although threat modeling has become a viable practice for secure software development, security testing with implicit and informal threat models has very limited ability to automatically generate security tests (Lijo Thomas). The Predicate/ Transition (PrT)

© 2013 IJAIR. ALL RIGHTS RESERVED

netsare used for automated security testing by using formal threat models represented. PrT nets are high-level Petri nets, which are a well-studied mathematically-based method for modeling and verifying distributed systems. To achieve secure design, wehave used PrT nets as a unified formalism for modelingsystem functions, security threats, and security features.Presence (or absence) of the threats can be verified againstthe functions before (or after) the security features areapplied. Recently, we have implemented an animator forstepwise simulation of attack behaviors. Based on this work,this paper aims at automated security testing with PrT net-based threat models.

Our developed model will acquire the agent's guilt. To identify the leaker of whom we sent the objects, we present the algorithms also. We also add the fake objects to the original objects. Such objects will appear related to the agents but do not correspond to real entities. The fake objects will act as a watermark for the entire set, without modifying any individual members. If it turns out that an agent was given one or more fake objects that were leaked, then the distributor can be more confident that agent was guilty.

Section 2 explains our problem setup and the used notation. The 4 & 5Sections will represent a model for calculating data leakage "guilt" probabilities. The data allocation strategies will be explained in Sections 6 and 7. Section 8 will explains the way we evaluate the strategies to identify the leaker in different data leakage scenarios.

2. PROBLEM SETUP AND NOTATION

2.1 Entities and Agents

A district but or owns a setT¼ft1;...; tm of valuable data objects. The set of agents U1;U2;…;Un, the distributor wants to share some of the objects, but leakage of objects to the other third parties are not interested. The objects in T could be of any type and size, e.g., they could be tuples in a relation, or relations in a database.

In (Panagiot is Papadimitriou)Ri T is a subset of objects is received by agent Ui. determined either by a sample request or an explicit request:

> Sample request Ri ¼SAMPLEðT; miÞ: Any subset of mi records from T can be given to Ui
>
> Explicit request Ri ¼EXPLICITðT; Cond iÞ: Agent Ui receives all To bisects that satisfy Cond i

For example: Say thatTcontains customer records for a given companyA. One company C1hires a marketing agency M1 to do an online survey of customers. If any customer wants to do the survey,agency M1 requests a sample of 1,000 customer records. At the same time, company C1 subcontracts with agent M2 to handle billing for all Mumbai customers. Thus, M2receives all T records that satisfy the condition "state is Mumbai." Our model for a sample of object requests can easily be extended to satisfy a condition (e.g., an agent wants any 100 Mumbai customer records). Also note that we need not concern ourselves with the randomness of a sample.

2.2 Guilty Agents

T is a set of objects leaked from agents is discovered by the distributor after distribution of objects. The third party called the target has been caught in jump to S. For example, the target website displays that it's Son the main website, or perhaps as part of a legal discovery process, the target turned over S to the distributor. Based on this we can say that the data was leaked because the agentsU1;...; Un have some of the data. And data S were obtained through other way by the target. For example, (Panagiot is, 2011) X is the customer S in the objects, and also a customer of some other company, and that company provided the data to the target.

Our aim is to found that the leaked data came from the agents compare to other sources. Intuitively, the agents may argue that they dint leak anything to third party is very hard. Similarly, it is hard to get that the objects obtained by other means to the targeted. Our aim is not only to estimate the likelihood of leaked data from the agents, but also like to find out the leaker in particular more likely. For the moment of time, in S object one is only given to the agentU1, and the other objects were given to all other agents, we may doubt onU1 more. The model we present next captures this intuition. We can say that an agent Ui is guilty due to his contribution on the objects leakage to the target. The agent Ui is guilty by Gi and the event that agent Ui is guilty for leakage of set S by Gi jS. The next step is to estimate Prf Gi jSg, i.e., the probability that agent Ui is guilty given evidence S.

RELATED WORK

Our proposed detection of guilt approachis related to the data provenance problem [2]: tracing the lineage of S objects implies essentially the detection of the guilty agents. Tutorial [3] provides a good overview on the research conducted in this field. Suggested solutions are domain specific, such as lineage tracing for data ware-houses [4], and assume some prior knowledge on the way a data view is created out of data sources. Our problem formulation with objects and sets is more general and simplifies lineage tracing, since we do not consider any data transformation from Ri sets to S.

How many times the data allocation strategies are concerned, our work is relevant to watermarking technique will establish original ownership of distributed objects. Watermarks were initially used in images [5], video [6], and audio data [6] whose digital representation includes considerable redundancy. Our approach is similar as watermarking that provides agents some receiver identifying information. By its nature, a watermark modifies the item being watermarked. If the object is watermarked it cannot be modified, and another watermark cannot be inserted. In such cases, attaching watermarks to the distributed data are not applicable.

Other works are also there to allow authorized user access control to access sensitive data through access control policies. Those approaches will prevent in sense the data leakage by sharing information only with trusted parties. However, these policies impossible to satisfy agents' requests.

The main focus in this paper is on automated generation of executable security test code from Threat Model-Implement Description (TMID) specifications. A TMID specification consists of a threat model (i.e., PrT net) and a Model-Implementation Mapping (MIM) description. A threat model describes how toattack to violate a security goal. A MIM description maps the individual elements of a threat model to their implementation constructs. Given a TMID specification, our approach can generate all attack paths from the threat model and then convert them into executable code according to the MIM description. As such, the security tests generated from the threat model can be executed automatically. We have implemented our approach in Integration and System Test Automation (ISTA), a framework for automated test code generation from PrT nets [12]. Currently, ISTA uses either HTML/Selenium IDE or C as the target language of test code for generating security tests from TMID specifications. Selenium 2 is a Firefox plug-in for creating, recording, and replaying test cases for web applications.

We have built comprehensive threat models according to the threat classification system STRIDE (spoofing identity, tampering with data, repudiation, information disclosure, denial of service, and elevation of privilege) [10], [13]. STRIDE has been widely used for threat modeling [46]. The security tests generated from the threat models of these systems have revealed security vulnerabilities and risks in each system. While all attack paths are generated automatically from the threat models, about 95 percent of them are converted to executable test code and can be performed automatically. To further evaluate the vulnerability detection capability of the security tests, we have applied them to the security mutants of the above systems. Each mutant is a variation of the original version with one vulnerability injected deliberately. A mutant is said to be killed if at least one of the security tests is a successful attack against the mutant.

Our experiments shows that the security tests generated from the security models are very effective they have killed about 90 percent of the mutants. The contribution of this paper is twofold. First, our

© 2013 IJAIR. ALL RIGHTS RESERVED

approach can generate executable security tests from rigorous threat models that capture various security attacks, such as spoofing, tampering with data, information disclosure, denial of service, and elevation of privilege. It is recognized that security testing of software applications needs to be performed from the adversary’s perspective, i.e., how the adversary might attack the system under test (SUT).

The existing security testing techniques primarily use implicit threat models (e.g., thoughts in security tester’s mind) or informal threat descriptions (e.g., represented by attack trees). However, security testing with informal threat specifications (e.g., attack trees) has very limited ability to automate test generation or test execution [14], [15]. In this paper, threat modeling is based on a rigorous formalism, PrT nets, from an effective approach to secure software design [45]. By using PrT nets to model system functions, security threats, and security features, presence (and absence) of the security threats can be verified against the system functions before (and after) the security features are applied. Threat models resulted from such a design process can be leveraged to generate security tests for validating the resultant implementation. In addition, the existing research on model-based testing has focused on test generation from intended behavior models [16], not from rigorous threat models. Second, we used security mutation (i.e., injection of various security vulnerabilities) for evaluating the effectiveness of our approach.

Traditional mutation testing research focuses on fault injection by making syntactic changes to a target program or specification [13], such as modification of && (and) to jj (or) in a condition. Obviously, such mutants are unlikely security vulnerabilities because they have not taken the semantics into consideration. The existing work on security mutation analysis focuses on vulnerability injection for particular types of attacks (e.g., injection, XSS, and buffer over flow) and fault injection for role-based access control (RBAC) policies [16].

THREAT MODELS FOR SECURITY TESTING

This section introduces TMID, the front-end input language for automated security testing. A TMID specification includes a threat model and a MIM specification. A threat model describes how attacks can be performed against the SUT, whereas a MIM specification maps the elements of a threat model to implementation-level constructs. The for merits used to generate security tests and the latter is used to convert them into executable code.

Threat Models:

Definition 1 (PrT net).A PrT net N is a tuple <P;T;F;I;P;L; ’;M0>, where1. Pis a set of places (i.e., predicates), T is a set of transitions, F is a set of normal arcs, and Iis set of inhibitor arcs.

2.P is a set of constants, relations (e.g., equal to and greater than), and arithmetic operations (e.g., addition and subtraction).

3. L is a labeling function on arcs F[I. L(f) is a label for arc f. Each label is a tuple of variables and/or constants inP

4. ’is a guard function on T:’ðtÞ;t’s guard condition, is built from variables and the constants, relations, and arithmetic operations inP.5. M0¼Sp2PM0ðpÞis an initial marking, hereM0ðpÞ is the set of tokens in place p. Each token is a tuple of constants in P.

A simplified version of traditional PrT nets [8]. This formalism has been applied successfully to threat modeling in a formal method for secure software design [15]. It is also supported by an efficient verification technique [18].Suppose each variable starts with a lower case letter or question mark (?) and each constant starts with an uppercase letter or digit.<6c>denotes the zero-argument tuple for a token or default arc label if an arc is not labeled.pðV1;...;VnÞ denotes token<V1;...;Vn>in place p. Places and transitions are represented by circles and rectangles ,respectively. An arrow represents a normal arc; a line segment with a small solid diamond on both ends represents an inhibitor arc. Fig. 1 shows an example.

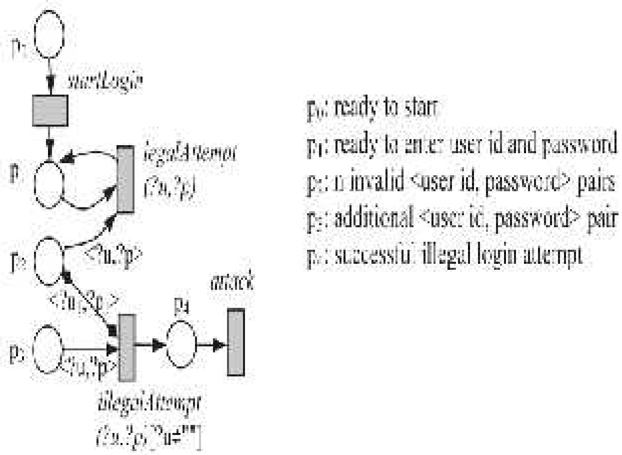
p0
startLogin
p1
legalAttempt
(?u,?p)
p2
<?u,?p>
<?u1,?p1>
p3
<?u,?p>
p4
attack
illegalAttempt
(?u,?p)[?u≠""]
p0: ready to start
p1: ready to enter user id and password
p2: n invalid <user id, password> pairs
p3: additional <user id, password> pair
p4: successful illegal login attempt

Fig. 1. PrT net for a dictionary attack.

Transitions legal Attempt and illegal Attempt have formal parameters ð?U;?pÞ. Illegal Attempt also has a guard condition? u6¼0000. Let p be a place and t be a transition. P Is called an input(or output) place of t if there is a normal arc from p to t(or from t to p). p is called an inhibitor place if there is an inhibitor arc between p and t. Let? x=V be a variable binding, where ?x is bound to value V. A substitution is a set of variable bindings. In substitution f?u=ID1;?p=PSWD1g, ?u and ?p are bound to ID1 and PSWD1,respectively. Let be a substitution andlbe an arc label .l= denotes the tuple (or token) obtained by substituting eachvariable inl for its bound value in .Ifl¼<?u;?p>and¼f?u=ID1;?p=PSWD1g,thenl=¼<ID 1;PSWD1>.

Transition t is said to be enabled or firable by under a marking if 1) each input placep of t has a token that matches l=, where l is the normal arc label from p to t;2) each inhibitor placep of t as no token that matches l=,where l is the inhibitor arc label; and 3) the guard condition of t evaluates to true according to.

Suppose M0¼fp1;p2ðID1;PSWD1Þ;p3ðIDn þ1;PSWDnþ1Þgfor the net in Fig. 1. Legal Attempt is enabledby¼f?u=ID1;?p=PSWD1gbecausep1 has a token (i.e.,<6c>) and p2 have a token<ID1;PSWD1>that matches<?u;?p>=. Illegal Attempt is not enabled under M0 becausep2, as an inhibitor place, has a token that can be unified with the arc label <?u1;?p1>. Inhibitor arcs represent negation.

Firing an enabled transition t with substitution underM0 removes the matching token from each input place and adds new token l=to each output place, where l is the arc label from t to the output place. This leads to a new markingM1. Firing tð?x1;...;?xnÞ ith¼f?x1=V1;...;?xn=Vngisdenoted bytor tðV1;...;VnÞ:M0;t11;M1...tnn;Mn,orsimply t11;...;tnn, is called a firing sequence, whereetið1inÞis a transition, ið1inÞis the substitutionfor firingti, and Mi ð1inÞis the marking after tifires ,respectively. A marking M is said to be reachable from M0if there is such a firing sequence that transforms M0 to M. Note that evaluation of a guard condition for transition firing may involve comparisons,arithmetic operations, andbinding of free variables to values. For example, evaluationofz¼xþ1wherexis bound to two will first computexþ1and then bind z to three. Therefore, a firing sequence can imply a sequence of data transformations.

Definition 2 (Threat model or net).A PrT net<P;T;F;I;P;L;';M0>is a threat model or net if This one or more attack transitions (suppose the name of each attack transition starts with "attack"). The firing of an attack transition is a security attack or a significant sign of security vulnerability.

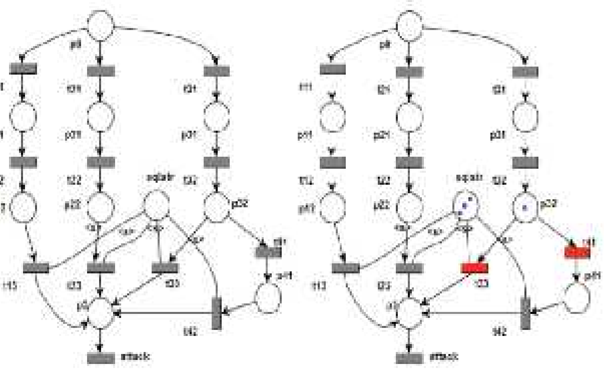

Fig 2. A threat net for SQL injection attacks.

The net in Fig. 2 models a dictionary attack against a system that allows only n invalid login attempts for authentication. It describes that the adversary tries to makesnþ1login ttempts.p2holdsninvalid<user id; password>pairs andp3 holds one invalid<user id;

© 2013 IJAIR. ALL RIGHTS RESERVED

password>pair.SupposeM0¼fp0;p2ðID1;PSWD1Þ;p2ðID2;PSWD2Þ;p2ðID3;PSWD3Þ;p3ðIDnþ1;PSWDnþ1Þg:Then, the following firing sequence violates the authentication policy of a system that allows only three invalid login attempts:

Result Analysis

Both case studies have used a structured process to build threat models by applying the STRIDE classification to the system functions. STRIDE helps identify threats to all security goals, including confidentiality, integrity, avail-ability, authentication, authorization, and non-repudiation.

The threat nets for each case study have covered all systems functions and threat types. Through threat modeling, a security tester can gain an in-depth understanding about the SUT. Threat models document the tester's thoughts on the goals and processes of security attacks. This is critical to effective security testing. To achieve an attack goal, a real-world adversary may only need one or few ways to break into the system. Security testing, however, must consider as many potential attacks as possible.

In both studies, attack paths are all generated automatically from the threat nets. Majority of them are successfully converted into executable code in that the MIM specifications can be developed. Whether the MIM for a threat net can be specified depends on whether the individual actions and conditions are programmable. 95.1(98/103) and 94.7 percent (72/76) of the tests can be fully or partially automated for Magento. Respectively. Although the prior work on testing with attack trees [18] can generate attack paths automatically, these attack paths are usually ambiguous because the attack actions and conditions originated from the attack trees are described in plain text. Transformation of the attack paths to executable tests can only be done by hand. In comparison, our new approach allows for a high degree of automation for security testing.

Both studies show that security testing with for mal threat models is very effective. The security tests have found vulnerabilities in each system. They have killed 88.9(56/63) and 92.1 percent (35/38) of the security mutants, respectively. The overall vulnerability detection rate is91=101¼90percent. The main reason for the effectiveness is that the threat models are built as if the tester were an intelligent adversary. The tests generated from the threat models are directly on the target, i.e., vulnerabilities that can be exploited by attacks. Nevertheless, testing with threat models also has limitations. It is difficult to reveal vulnerabilities that are not anticipated by the threat models.

DATA ALLOCATION PROBLEM

Our main focus in this paper is problem in data allocation: the way the distributor can" intelligently" distribute the data to the agents to improve the chances of finding a guilty agent? Fig. 2 explains that the problem address four instances, depending on the agents request on data and whether "fake objects" are allowed .We handle two types of requests here were defined in Section 2: sample and explicit. The Fake objects are generated by the distributor that is not in set T. The fake objects are designed to look like real objects, and are distributed to agents together with T objects,(A kshy) in order to increase the chances of detecting agents that leak data.

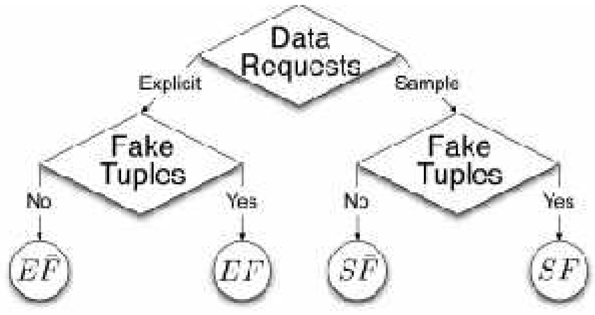

Fig. 3 Leakage problem instances

Fig. 3 shows that the representation of our four problem instances with the names EF, EF, SF, and SF, where E stands for explicit requests, S for sample requests, F for the use of fake objects, and F for the case where fake objects are not allowed.

For simplicity, we assume that problem instances in E, all agents' requests are done by explicitly, while in the instances S, all requests are sample. We extended this to handle mixed cases also, with some explicit and sample requests. For example how mixed case handled, but not elaborates further. Assume that we have two agents with requests R1¼EXPLICITðT; cond1Þ andR2¼SAMPLEðT0; 1Þ, whereT0 ¼EXPLICITðT; cond2Þ.

Further, say thatcond1 is “state¼CA” (objects have a statefield). If agentU2has the same conditioncond2¼cond1, wecan create an equivalent problem with sample data requests on set T0. That is, our problem will be how to distribute theCA objects to two agents, withR1¼SAMPLEðT0; jT0jÞand R2¼SAMPLEðT0; 1Þ.If insteadU2 uses condition“state¼NY,” we can solve two different problems for setsT0 and T1. In each problem, we will have only one agent. Finally, if the conditions partially overlap.

For distributed data also the distributor will add fake objects it improves effectiveness in detecting guilty agents. However, fake objects may impact the correctness of what agents do, so they may not always be allowable.

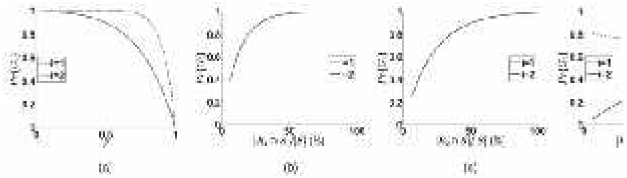

Fig. 4 Guilt probability as a function of the guessing probability p(a) and the overlap between SandR2 (b)-(d). In all scenarios, it holds thatR1\S¼SandjSj¼16. (a)jR2\SjjSj ¼0:5, (b) p¼0:2, (c) p¼0:5, and (d) p¼0:9.

To detect leakage of data is not a new idea called the perturbing. However, in most cases, individual objects are perturbed, e.g., by adding random noise to sensitive salaries, or adding a watermark to an image. In our case, we are perturbing the set of distributor objects by adding Fig. 4. Guilt probability as a function of the guessing probabilityp(a) and the overlap betweenSandR2 (b)-(d). In all scenarios, it holds thatR1\S¼SandjSj¼16. (a)JR2\SjjSj ¼0:5, (b) p¼0:2, (c) p¼0:5, and (d) p¼0:9.

ALLOCATION STRATEGIES

This section illustrates the allocation strategies those involved to solve exactly or approximately the scalar versions of (8) for the different instances presented in Fig. 3. Where we cannot solve the optimization problem there we place the approximate solutions. The proofs of theorems that are stated in the following sections are available in [14].

Explicit Data Requests

In EF class problems the distributor is not allowed to add fake objects to the distributed data. So, the data allocation is fully defined by the agents’ data requests. Therefore, there is nothing to optimize.

The objective values are initialized in EF problems, by agents’ data requests. Say, for example, that T={t1; t2}and there are two agents with explicit data requests such that R1={t1; t2} and R2={t1}. The value of the sum-objective is in this case the distributor cannot remove or alter theR1 orR2 data to decrease the overlap R1\R2. However, say that the distributor can create one fake object (B=1) and both agents can receive one fake object (b1=b2=1). In this case, the distributor can add one fake object to eitherR1 orR2 to increase the corresponding denominator of the summation term. Assume that the distributor creates a fake object f and he gives it to agentR1. Agent U1 has nowR1={t1; t2; f}andF1=ffgand the value of the sum-objective decreasesto13þ11¼1:33<1:5.

If the distributor is able to create more fake objects, he could further improve the objective. We present in Algorithms 1 and 2 a strategy for randomly allocating fake objects. Algorithm 1 is a general “driver” that will be used by other strategies, while Algorithm 2 actually performs the random selection. We denote the combination of Algorithm 1 with 2 random. We will use e-random as our baseline in our comparisons with other algorithms for explicit data requests.

Algorithm 1.Allocation for Explicit Data Requests (EF)

Input: R1;…;Rn, cond1;…; condn, b1;…;bn, B

Output: R1;…;Rn, F1;…;Fn

1: R; .Agents that can receive fake objects

```
fori¼1;…;ndo
if bi >0then
R R[fig
Fi ;
whileB>0do
i SELECTAGENTðR;R1;…;RnÞ
f CREATEFAKEOBJECTðRi;Fi; cond iÞ
Ri  Ri [ffg
Fi  Fi [ffg
bi  bi ————————————————1
if bi ¼0then
R RnfRig
B B————————————————1
```

EXPERIMENTAL RESULTS

Our allocation algorithms are implemented in Python and we conducted experiments with simulated data leakage problems to evaluate their performance. In Section 8.1, we present the metrics we use for the algorithm evaluation.

Metrics

we presented algorithms to optimize the problem of (8) that is an approximation to the original optimization problem of (7). In this section, we evaluate the presented algorithms with respect to the original problem. In this way, we measure not only the algorithm performance, but also we implicitly evaluate how effective the approximation is. The objectives in (7) are the difference functions.

We evaluate a given allocation with the following objective secularizations as metrics: Metric is the average of ði; jÞ values for a given allocation and it shows how successful the guilt detection is, on average, for this allocation. For example, if it 0:4, then, on average, the probability Prf GijRig for the actual guilty agent will be 0.4 higher than the probabilities of non-guilty agents. Note that this scalar version of the original problem objective is analogous to the sum-objective scalarizations of the problem of (8). Hence, we expect that an algorithm that is designed to minimize the sum-objective will maximize. Metric Min is the minimum ði; jÞ value and it corresponds to the case where agent Ui has leaked his data and both Ui and another agent Uj have very similar guilt probabilities. If min is small, then we will be unable to Identify Ui as the leaker, versus Uj. If min is large, say, 0.4,then no matter which agent leaks his data, the probability that he is guilty will be 0.4 higher than any other non-guilty agent. This metric is analogous to the max-objective scalarizations of the approximate optimization problem.

The selected values for these metrics are depending on the application. In particular, they depend on what might be considered high confidence that an agent is guilty. For instance, say thatPrfGijRig¼0:9 is enough to arouse our suspicion that agent Ui leaked data. Furthermore, say that the difference between PrfGijRig and any other PrfGjjRig is at least 0.3.In other words, the guilty agent isð0:90:6Þ=0:600%¼50%more likely to be guilty compared to the other agents. In this case, we may be willing to take action against Ui. In the rest of this section, we will use value 0.3 as an example of what might be desired in values.

To calculate the guilt probabilities and differences, we use throughout this section p=0:5. Although not reported here, we did the experiments with other p values and observed that the relative performance of our algorithms and our main conclusions do not change. If p approaches to 0, it becomes easier to find guilty agents and algorithm performance converges. On the other hand, ifpapproaches1, the relative differences among algorithms grow since more evidence is needed to find an agent guilty.

Explicit Requests

In the first place, the goal of these experiments was to see whether fake objects in the distributed data sets yield significant improvement in our chances of detecting a

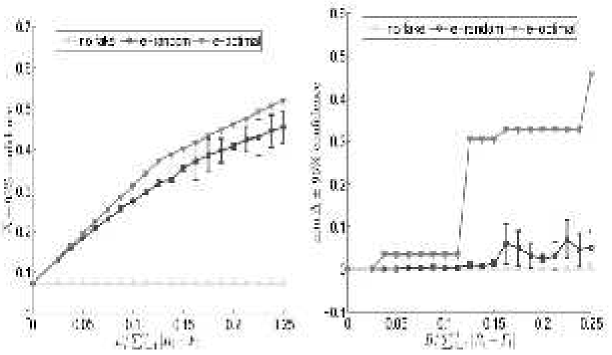

Fig. 5 Evaluation of explicit data request algorithms
(a) Average, (b) Average min

We focused on with few objects that are shared among multiple agents. These are the most interesting scenarios, since object sharing makes it difficult to distinguish a guilty from non-guilty agents. A scenario with more objects to distribute or shared among fewer agents are obviously easier to handle. As far as scenarios with many objects to distribute and many over-lapping agent requests are concerned, they are similar to the scenarios we study, since we can map them to the distribution of many small subsets.

In our scenarios, we have a set of jTj¼10objects for which there are requests byn¼10 different agents. Our assumption is that the each agent requests 8 particular objects out of these 10. Hence, each object is shared, on average, among¼8agents. This scenarios yield more similar that the agent guilt probabilities and it is most important to add fake objects. We generated a random scenario thatyielded¼0:073andmin¼0:35and we applied the algorithms e random and e-optimal to distribute fake objects to the agents. The number B varied with distributed fake objects from 2 to 20, and for each value of B, we ran both algorithms to allocate the fake objects to agents. We rane-optimal once for each value of B, since it is a deterministic algorithm. Algorithm e- random is randomized and we ran it 10 times for each value of B. The results we present are the average over the 10 runs. Fig. 3a shows how fake object allocation can affect.

There are three curves in the plot. The solid curve is constant and shows the value for an allocation without fake objects (totally defined by agents' requests). The other two curves look at algorithms e-optimal and e-random. The X-axis and the Y-axis shows the ratio between the numbers of distributed fake objects to the total number of objects that the agents explicitly request.

We observe that distributing fake objects can significantly improve, on average, the chances of detecting a guiltyagent. The random allocation yields > 0.3 for approximately 10 to15 percent fake objects. The use of e-optimalimprovesfurther, since thee-optimalcurve is >95 % consistently in intervals ofe-random. If the agent dint require same number of objects theperformance difference between the two algorithms wouldbe greater, since this symmetry allows non smart fake object scenarios. However, we do not study more this issue here,since the advantages ofe-optimalbecome obvious when welook at our second metric.

The function of the fraction of fake objects. The insignificant improvement in random allocation shows the plot chances of detecting a guilty agent in the worst-case scenario. This was expected, since e-random does not take into consideration due to that each agent "must" receive a fake object to differentiate their requests from other agents. On the contrary, algorithm e-optimal can yield min >0:3with the allocation of approximately 10 percent fake objects. This improvement is very important taking into account that without fake objects, values min and are close to 0.

By allocating 10 percent of fake objects, in worst case also the distributor can detect a guilty agent. Without allocating fake objects, the distributor was unsuccessful in the worst case as well as in average case also. Our e-optimal curve has two jumps due to the symmetry in our scenario. Our e-optimal algorithm allocates one fake object per each agent before allocating a second fake object to other agents.

CONCLUSION

The agents may leak sensitive data that may unknowingly or maliciously. And even if we want to handle the sensitivedata perfectly in this world we could do watermarking each object sothat we can trace its origins with absolute certainty.In many cases in this world, we must work with agentsthose are may not be 100 percent trusted, and we cannot confirm that the leaked object came from an agent or from other source, since our data cannot admit watermarks.

Automated generation of security test code largely depends on whether or not threat models can be formally specified, whether or not individual test inputs (e.g., attack actions with particular input data) and test oracles (e.g., for checking system states) can be programmed. A system that s designed for testability and traceability would certainly facilitate automating its security testing process. For example, threat models identified and documented in the design phase can be reused for security test generation.

Access or methods designed for testability (i.e., for accessing system states) are useful for verification of security test oracles. The traceability of design-level functions in the implementation can facilitate the mapping from individual actions in threat models to implementation constructs. It is worth pointing out that the threat models in our approach can be built at different levels of abstraction. They do not necessarily specify design-level security threats.

Software security is a complex problem; there is no silver bullet [17]. Different techniques are often needed in order to achieve a high level of security assurance. In particular, testing for security and static analysis for security are two different approaches. It is of interest to conduct a comparative study on their cost effectiveness.

REFRENCES

[1] R. Agrawal and J. Kiernan, "Watermarking Relational Databases,"Proc. 28th Int'l Conf. Very Large Data Bases (VLDB '02),VLDBEndowment, pp. 155-166, 2002.

[2] C. Bezemer, A. Mesbah, and A. van Deursen, "Automated Security Testing of Web Widget Interactions,"Proc. Seventh Joint Meeting of the European Software Eng. Conf. and the ACM SIGSOFT Symp. Foundations of Software Eng. (ESE/FSE '09),pp. 81-90, 2009

[3] P. Buneman, S. Khanna, and W.C. Tan, "Why and Where: ACharacterization of Data Provenance,"Proc. Eighth Int'l Conf.Database Theory (ICDT '01),J.V. den Bussche and V. Vianu, eds.,pp. 316-330, Jan. 2001.

[4] P. Buneman and W.-C. Tan, "Provenance in Databases,"Proc.ACM SIGMOD,pp. 1171-1173, 2007.

[5] R. Chandramouli and M. Blackburn, "Automated Testing of Security Functions Using a Combined Model & Interface Driven Approach,"Proc. 37th Hawaii Int'l Conf. System Sciences,pp. 299-308, 2004

[6] S. Czerwinski, R. Fromm, and T. Hodes, "Digital Music Distribu-tion and Audio watermarking," http://www.scientificcommons.org/43025658, 2007.

[7] J. Fonseca, M. Vieira, and H. Madeira, "Testing and Comparing Web Vulnerability Scanning Tools for SQL Injection and Xss Attacks,"Proc. 13th Pacific Rim Int'l Symp. Dependable Computing, pp. 365-372, Dec. 2007

[8] F. Hartung and B. Girod, "Watermarking of Uncompressed andCompressed Video,"Signal Processing,vol. 66, no. 3, pp. 283-301,1998.

[9] P. Hope and B. Walther,Web Security Testing Cookbook: Systematic Techniques to Find Problems Fast.O' Reilly Media, Inc., 2009.

[10] M. Howard and D. LeBlanc,Writing Secure Code,second ed. Microsoft Press, 2003

[11] B. Mungamuru and H. Garcia-Molina, "Privacy, Preservation andPerformance: The 3 P's of Distributed Data Management,"technical report, Stanford Univ., 2008.

[12] V.N. Murty, "Counting the Integer Solutions of a Linear Equationwith Unit Coefficients,"Math. Magazine,vol. 54, no. 2, pp. 79-81,1981.

[13] Y. Jia and M. Harman, "An Analysis and Survey of the Development of Mutation Testing,"IEEE Trans. Software Eng., vol. 37, no. 5, pp. 649-678, Sept./Oct. 2011.

[14] J. Julliand, P.A. Masson, and R. Tissot, "Generating Security Tests in Addition to Functional Tests," Proc. Third Int'l Workshop Automation of Software Test,pp. 41-44, 2008

[15] J.J.K.O. Ruanaidh, W.J. Dowling, and F.M. Boland, "Watermark-ing Digital Images for Copyright Protection," IEE Proc. Vision,Signal and Image Processing,vol. 143, no. 4, pp. 250-256, 1996.

[16] R. Sion, M. Atallah, and S. Prabhakar, "Rights Protection forRelational Data,"Proc. ACM SIGMOD,pp. 98-109, 2003.

[17] J. Kong, D. Xu, and X. Zeng, "UML-Based Modeling and Analysis of Security Threats,"Int'l J. Software Eng. and Knowledge Eng., vol. 20, no. 6, pp. 875-897, Sept. 2010.

[18] L. Sweeney, “Achieving K-Anonymity Privacy Protection UsingGeneralization and Suppression,” http://en.scientificcommons.org/43196131, 2002

[19] D. Xu, “Software Security,”Wiley Encyclopedia of Computer Science and Engineering ,B.W. Wah, ed., vol. 5, pp. 2703-2716, John Wiley & Sons, Inc., Jan. 2009.

© 2013 IJAIR. ALL RIGHTS RESERVED